Car Damage Detection: A Comparative Analysis of Two Models

Team: Roshni V (CB.EN.U4CYS21061)

Mentor: Mr. Ramaguru Radhakrishnan

1) Problem Statement:

The objective of this project is to develop a car damage detection system using deep learning techniques. The aim is to compare two different models and evaluate their performance in detecting and classifying car damage. The project utilizes TensorFlow and Jupyter Notebook for implementing the code, and the dataset used consists of 1,920 images obtained from Kaggle. Three research papers, namely "Car Damage Detection and Classification," "Car Damage Assessment Based on VGG Models," and "A Very Deep Transfer Learning Model for Vehicle," serve as references for the project.

The project aims to help out with Car Insurance claims for,

1) Damage Detection,(binary) and

2) Damage Localization(multiclass)

2) Achievements:

a) Model 1: Utilizing TensorFlow, NumPy, Pandas, and OS:

Preprocessing: Loading and converting images to arrays, applying preprocessing techniques to both the training and test data, and appending corresponding labels.

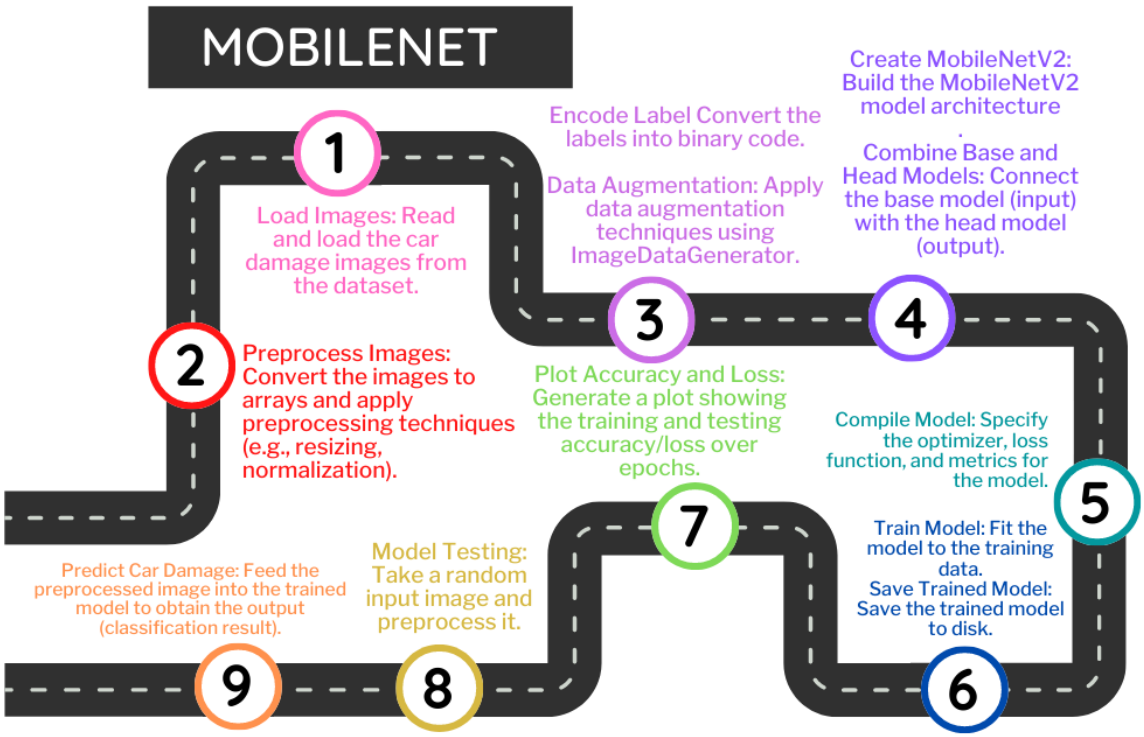
Encoding Labels: Converting labels and data into binary code.

Data Augmentation: Implementing ImageDataGenerator for augmenting the data.

MobileNetV2: Creating a model by combining a base model with a head model, allowing for classification.

Training and Testing: Compiling the model with an optimizer, fitting the model to the training data, and saving the trained model to disk.

Evaluation: Plotting training and testing accuracy/loss over epochs and assessing the model's performance.



b) Model 2: Utilizing TensorFlow, Keras, Scikit-learn, Imutils, NumPy, Matplotlib, and OS:

Preprocessing: Loading and converting images to arrays, applying preprocessing techniques to both the training and test data, and appending corresponding labels.

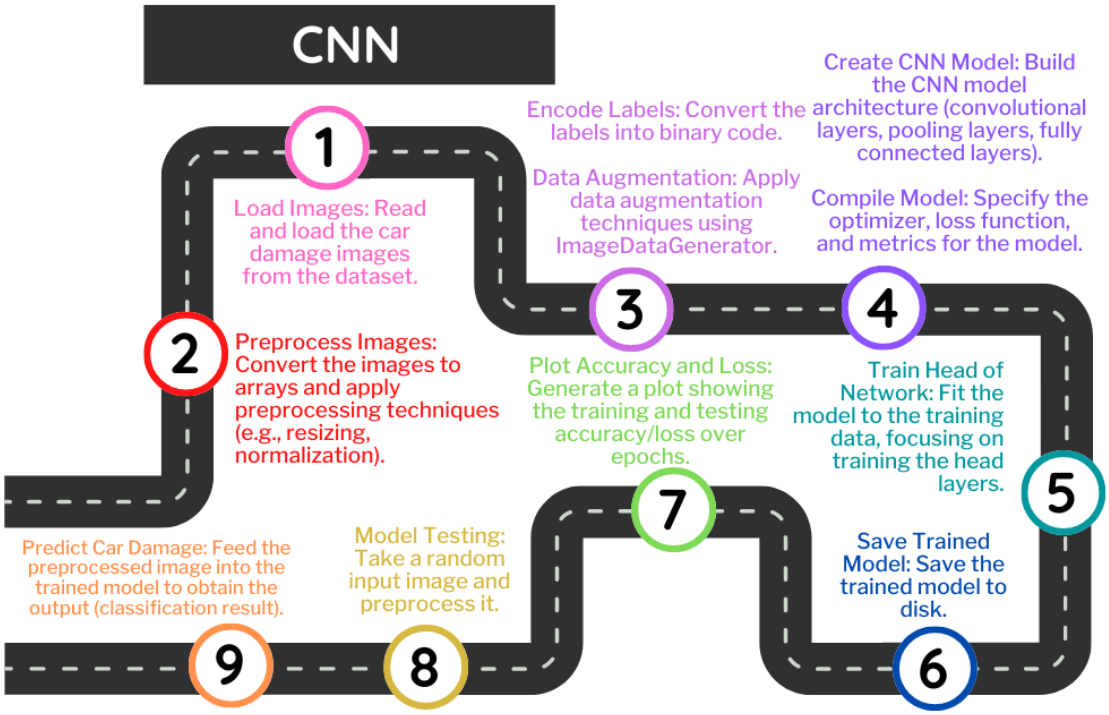
Encoding Labels: Converting labels and data into binary code.

Data Augmentation: Utilizing ImageDataGenerator for data augmentation.

CNN Model: Implementing a Convolutional Neural Network (CNN) for feature extraction and classification.

Training and Testing: Compiling the model with an optimizer, training the head of the network, saving the trained model to disk, and evaluating the model's performance.

Evaluation: Plotting training and testing accuracy/loss over epochs and assessing the model's performance.



3) Discussion of Results or Inferences:

Both models where put under tensorboard visualizer to tap individual outputs at the end of each layer. Both models were successful in detecting and classifying car damage. The comparative analysis between Model 1 and Model 2 reveals the following observations:

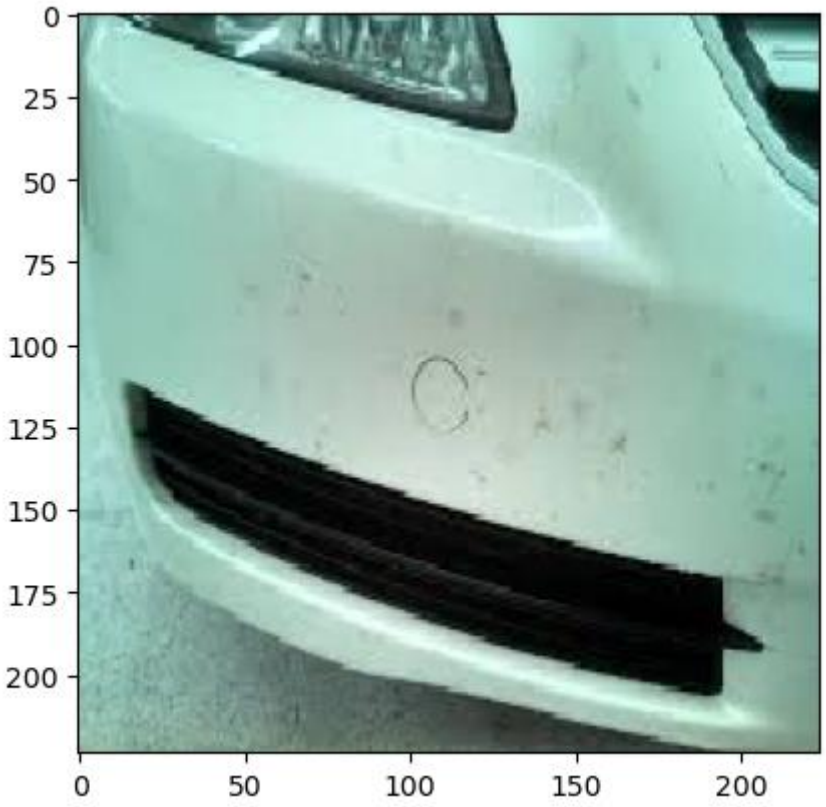
Model 1 employs the MobileNetV2 architecture, which is specifically designed for mobile and embedded devices, making it a lightweight and efficient solution.

Model 2 utilizes a CNN model with multiple convolutional layers and dropout regularization, providing more flexibility and customization options.

Both models achieved satisfactory accuracy and loss metrics during training and testing.

The choice of model depends on the specific requirements of the application, such as computational resources, deployment environment, and desired trade-offs between accuracy and model complexity.

4) MobileNet Model Summary:



Block 1:

block_1_conv (Conv2D)

block_1_bn (BatchNormalization)

block_1_relu (ReLU)

block_1_pad (ZeroPadding2D)

block_1_pool (MaxPooling2D)

Block 2:

block_2_expand (Conv2D)

block_2_expand_BN (BatchNormalization)

block_2_expand_relu (ReLU)

block_2_depthwise (DepthwiseConv2D)

block_2_depthwise_BN (BatchNormalization)

block_2_depthwise_relu (ReLU)

block_2_project (Conv2D)

block_2_project_BN (BatchNormalization)

block_2_pool (MaxPooling2D)

block_2_add (Add)

Block 3:

block_3_expand (Conv2D)

block_3_expand_BN (BatchNormalization)

block_3_expand_relu (ReLU)

block_3_pad (ZeroPadding2D)

block_3_depthwise (DepthwiseConv2D)

block_3_depthwise_BN (BatchNormalization)

block_3_depthwise_relu (ReLU)

block_3_project (Conv2D)

block_3_project_BN (BatchNormalization)

block_3_add (Add)

Block 4:

block_4_expand (Conv2D)

block_4_expand_BN (BatchNormalization)

block_4_expand_relu (ReLU)

block_4_depthwise (DepthwiseConv2D)

block_4_depthwise_BN (BatchNormalization)

block_4_depthwise_relu (ReLU)

block_4_project (Conv2D)

block_4_project_BN (BatchNormalization)

block_4_add (Add)

Block 5:

block_5_expand (Conv2D)

block_5_expand_BN (BatchNormalization)

block_5_expand_relu (ReLU)

block_5_depthwise (DepthwiseConv2D)

block_5_depthwise_BN (BatchNormalization)

block_5_depthwise_relu (ReLU)

block_5_project (Conv2D)

block_5_project_BN (BatchNormalization)

block_5_add (Add)

block_5_activation (ReLU)

Block 6:

block_6_expand (Conv2D)

block_6_expand_BN (BatchNormalization)

block_6_expand_relu (ReLU)

block_6_pad (ZeroPadding2D)

block_6_depthwise (DepthwiseConv2D)

block_6_depthwise_BN (BatchNormalization)

block_6_depthwise_relu (ReLU)

block_6_project (Conv2D)

block_6_project_BN (BatchNormalization)

block_6_add (Add)

Block 7:

block_7_expand (Conv2D)

block_7_expand_BN (BatchNormalization)

block_7_expand_relu (ReLU)

block_7_depthwise (DepthwiseConv2D)

block_7_depthwise_BN (BatchNormalization)

block_7_depthwise_relu (ReLU)

block_7_project (Conv2D)

block_7_project_BN (BatchNormalization)

block_7_add (Add)

Block 8:

block_8_expand (Conv2D)

block_8_expand_BN (BatchNormalization)

block_8_expand_relu (ReLU)

block_8_depthwise (DepthwiseConv2D)

block_8_depthwise_BN (BatchNormalization)

block_8_depthwise_relu (ReLU)

block_8_project (Conv2D)

block_8_project_BN (BatchNormalization)

block_8_add (Add)

Block 9:

block_9_expand (Conv2D)

block_9_expand_BN (BatchNormalization)

block_9_expand_relu (ReLU)

block_9_depthwise (DepthwiseConv2D)

block_9_depthwise_BN (BatchNormalization)

block_9_depthwise_relu (ReLU)

block_9_project (Conv2D)

block_9_project_BN (BatchNormalization)

block_9_add (Add)

Block 10:

block_10_expand (Conv2D)

block_10_expand_BN (BatchNormalization)

block_10_expand_relu (ReLU)

block_10_depthwise (DepthwiseConv2D)

block_10_depthwise_BN (BatchNormalization)

block_10_depthwise_relu (ReLU)

block_10_project (Conv2D)

block_10_project_BN (BatchNormalization)

block_10_add (Add)

Block 11:

block_11_expand (Conv2D)

block_11_expand_BN (BatchNormalization)

block_11_expand_relu (ReLU)

block_11_pad (ZeroPadding2D)

block_11_depthwise (DepthwiseConv2D)

block_11_depthwise_BN (BatchNormalization)

block_11_depthwise_relu (ReLU)

block_11_project (Conv2D)

block_11_project_BN (BatchNormalization)

block_11_add (Add)

Block 12:

block_12_expand (Conv2D)

block_12_expand_BN (BatchNormalization)

block_12_expand_relu (ReLU)

block_12_depthwise (DepthwiseConv2D)

block_12_depthwise_BN (BatchNormalization)

block_12_depthwise_relu (ReLU)

block_12_project (Conv2D)

block_12_project_BN (BatchNormalization)

block_12_add (Add)

Block 13:

block_13_expand (Conv2D)

block_13_expand_BN (BatchNormalization)

block_13_expand_relu (ReLU)

block_13_pad (ZeroPadding2D)

block_13_depthwise (DepthwiseConv2D)

block_13_depthwise_BN (BatchNormalization)

block_13_depthwise_relu (ReLU)

block_13_project (Conv2D)

block_13_project_BN (BatchNormalization)

Block 14:

block_14_expand (Conv2D)

block_14_expand_BN (BatchNormalization)

block_14_expand_relu (ReLU)

block_14_depthwise (DepthwiseConv2D)

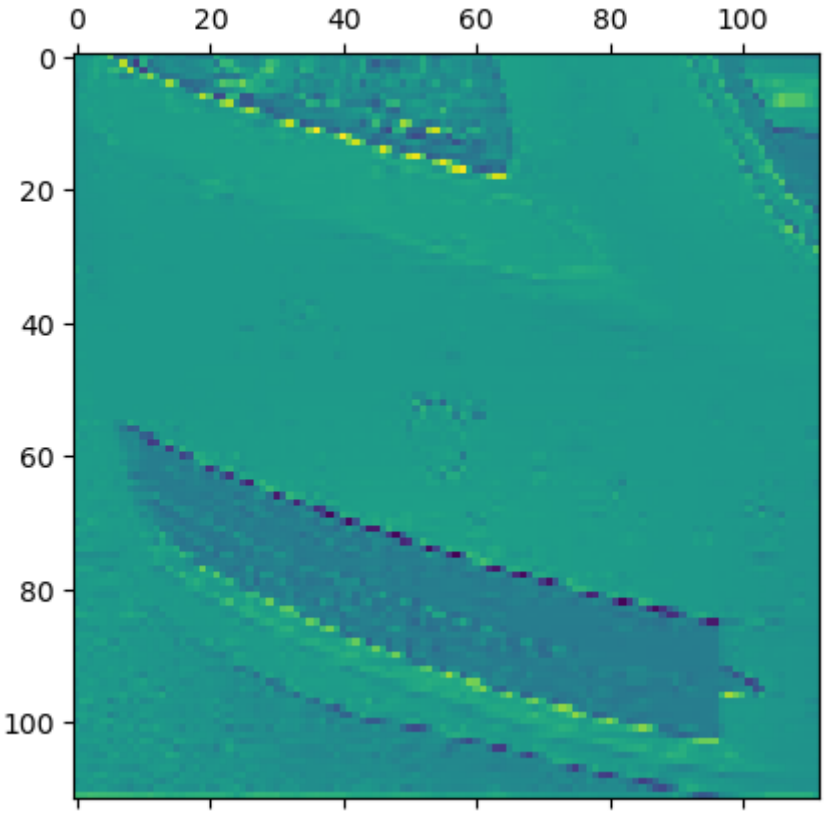
block_14_depthwise_BN (BatchNormalization)

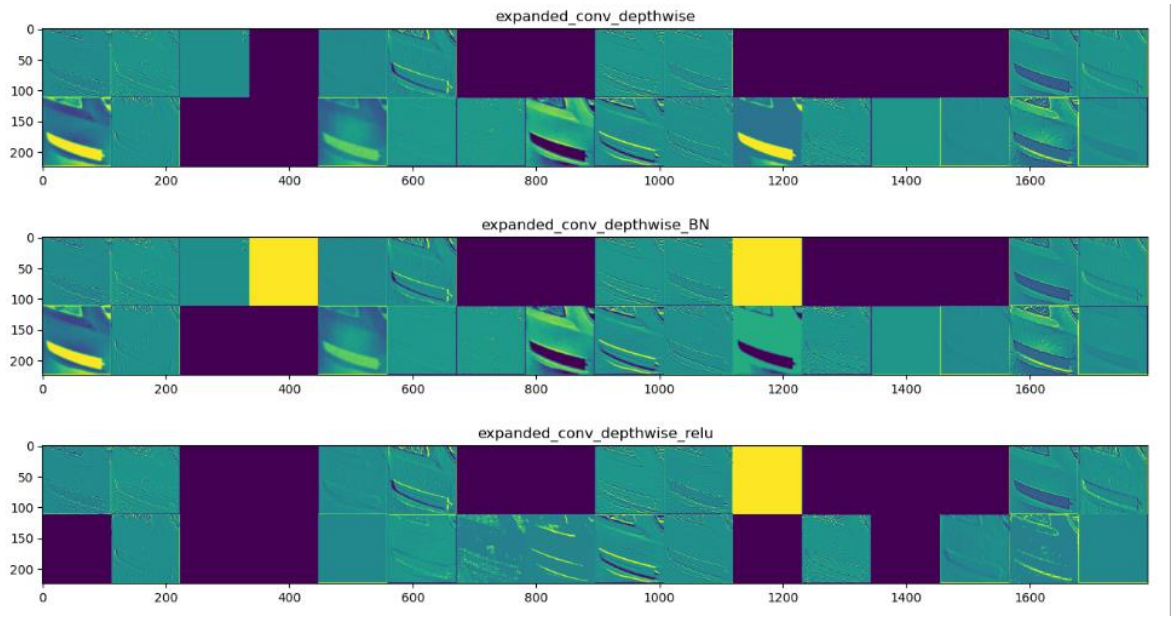
block_14_depthwise_relu (ReLU)

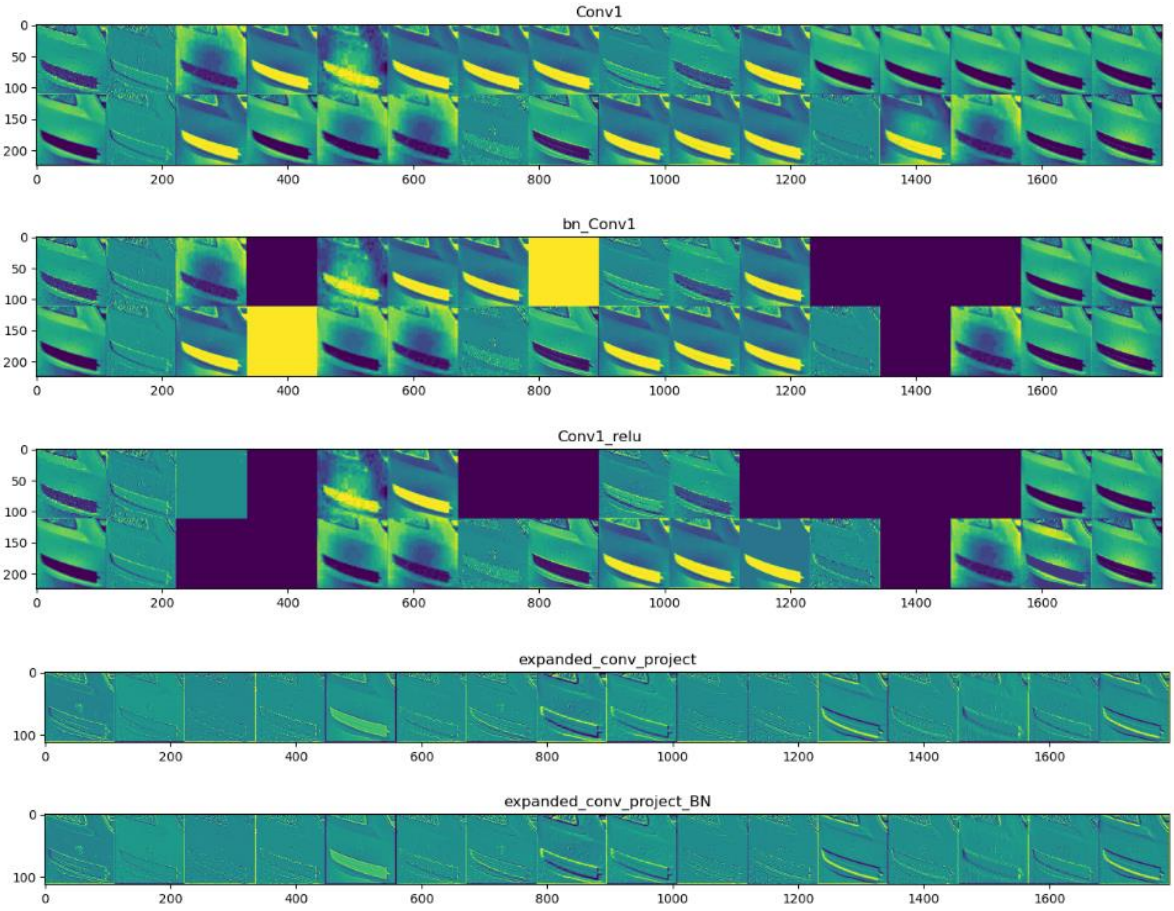
block_14_project (Conv2D)

block_14_project_BN (BatchNormalization)

block_14_add (Add)






Conv1
bn_Conv1
Conv1_relu
expanded_conv_project
expanded_conv_project_BN

5) Comparative Analysis:

The comparative analysis involves evaluating the performance of Model 1 (MobileNet) and Model 2 (CNN). The analysis includes metrics such as training and testing accuracy/loss, model complexity, computational resources required. The MobileNet model proved more efficient, with a maximum accuracy rancy of 78% to 85% and CNN model gave an accuracy of 55% to 70%.

*6) **Resources:***

1. **Dataset:** https://www.kaggle.com/datasets/anujms/car-damage-detection
2. **Research Paper:**
   a. Car Damage Detection and Classification: https://dl.acm.org/doi/10.1145/3406601.3406651
   b. Car Damage Assessment Based on VGG Models: https://site.ieee.org/thailand-cis/files/2019/12/JSCI8-Paper-5.pdf
   c. A Very Deep Transfer Learning Model for Vehicle: https://www.researchgate.net/publication/339763247_A_Very_Deep_Transfer_Learning_Model_for_Vehicle_Damage_Detection_and_Localization
3. **Raw Code:** https://www.kaggle.com/code/suyogmahagaonkar/car-damage-detection
4. **Updated Code**: https://github.com/CeraMapleheart/20CYS212_Car_Damage_Detection